IFIP TC8 Conference on
Governmental and Municipal Information Systems
Budapest, Hungary, September 8–11, 1987

NORTH-HOLLAND
AMSTERDAM · NEW YORK · OXFORD · TOKYO

GOVERMENTAL AND MUNICIPAL INFORMATION SYSTEMS

Proceedings of the IFIP TC8 Conference on
Governmental and Municipal Information Systems
Budapest, Hungary, September 8–11, 1987

edited by

Péter KOVÁCS

and

Elek STRAUB

1988

NORTH-HOLLAND
AMSTERDAM · NEW YORK · OXFORD · TOKYO

ISBN: 0 444 70377 2

Published by:
ELSEVIER SCIENCE PUBLISHERS B.V.
P.O. Box 1991
1000 BZ Amsterdam
The Netherlands

Sole distributors for the U.S.A. and Canada:
ELSEVIER SCIENCE PUBLISHING COMPANY, INC.
52 Vanderbilt Avenue
New York, N.Y. 10017
U.S.A.

LIBRARY OF CONGRESS
Library of Congress Cataloging-in-Publication Data

IFIP TC8 Conference on Governmental and Municipal Information Systems (1987 : Budapest, Hungary)
Governmental and municipal information systems : proceedings of the IFIP TC8 Conference on Governmental and Municipal Information Systems, Budapest, Hungary, September 8-11, 1987 / edited by Péter Kovács and Elek Straub.
p. cm.
Bibliography: p.
ISBN 0-444-70377-2 (U.S.)
1. Public administration--Data processing--Congresses. 2. Management information systems--Congresses. I. Kovács, Péter 1936- . II. Straub, Elek, 1944- . III. International Federation for Information Processing. Technical Committee for Information Systems. IV. Title.
JF1525.A8I448 1987
350'.00028'5--dc19 87-34954
CIP

PRINTED IN THE NETHERLANDS

PREFACE

Dear Reader,

The booklet which you now hold in your hands is another major undertaking by the IFIP TC8 (Information systems). The professionals working in TC8 have been attempting for some time to deploy the results achieved in informatics research and practice to all walks of economic and social life. For this purpose we have decided to organize in 1986 an open conference where we could discuss and get an overview of the special aspects pertaining to public administration informatics and its various areas of employment.

The conference was held in Budapest primarily because we wanted to meet in a region where informatics-supported decision making in public administration is considered a primary goal.

The conference was a great success in all respects, with a large attendance, and an extensive number of high-standard lectures presented. We had some 200 participants and more than 70 papers were given. The large number of lectures set the program committee a difficult task. Under the direction of Gösta Guteland the program committee had set as its goal to present the scope of public administration informatics in various aspects.

Besides the papers presented at the plenary meeting the lecturers spoke about their work in 5 different sections. These were divided into 5 subject areas as follows:

- Governmental information systems
- Development of information systems
- Local information systems and land data bank systems
- Office automation and development of networks
- Security and database management.

The common platform for the sectional meetings was outlined by P.A. Tas's lecture on "Information System Policy in Government". Mr. Tas's introductory paper suggestively and convincingly confirmed the existence of public administration informatics.

The substantial lectures and the interesting comments made by the audience contributed to the valuable summaries given at the closing session by the rapporteurs (Brussard, King, Ohashi, Schultz and Sutherland) whose task was to point out the major ideas and suggestions presented by each section.

Governmental and Municipal Information Systems
P. Kovács and E. Straub (Editors)
Elsevier Science Publishers B.V. (North-Holland)
IFIP, 1988

THE TIME BOMB

Knud Lysgaard

Kommunedata I/S, P.O. Box 720,
DK-9100 Aalborg.

It is a common practice to omit the century in dates. This article deals with the problems this causes in EDP systems as we approach the year 2000. In particular problems in large EDP installations are dealt with and there is a description of how Kommunedata I-S, Denmark, tackles this matter.

ADJUSTMENT OF TIME: YEAR 2000

In the early days of computer programming data storage was expensive and in many situations limited. Therefore, no superfluous information was stored. As almost all dates could be assumed to be in the 20th century there was no need to store the century. It became a habit to store dates with 2 digits for the year - just as it is a habit in many everyday situations only to note 2 digits, for example, 15/06-87.

The programmes to process the dates were, therefore, designed on the assumption, that the first two digits in all years are 19. At some point as we draw closer to year 2000 this assumption will, however, no longer qualify and programmes will either give wrong results - or simply refuse to produce results.

As almost all EDP systems store and process dates and as the majority of these dates are still stored and processed on the assumption that we are in the 20th century what we actually have here is a delayed-action time bomb ticking under our information society. If we do not act it will explode on 1st January 2000, at the latest - in many cases considerably earlier as systems are assumed to be able to handle dates forward in time.

NO HELP FROM SUPPLIERS

We have to do something. We cannot blame the problem on others than ourselves. Unfortunately, we cannot reckon on others solving it for us either.

It's true it is not only we users of EDP technology who have a problem. The EDP vendors have in many cases built up their operating systems on exactly the same weakness and for exactly the same reason: the high price and limited storage space. The EDP suppliers who wish to be on the market in 2000 will, of course,

contrive to solve their own problems. They will probably offer some tools which can assist the rest of us a little in solving our problems but there is no indication that the suppliers are able to contribute markedly to a solution of our problem.

CONVERSION OF FORMAT

If we had worked out all date formats with all 4 digits in the year it would not have been necessary to incorporate assumptions in our programmes and the turn of the century would neither have given us headaches nor sleepless nights.

What is more obvious than changing all 2 digit years to 4 digits?

From a technical point of view such a conversion is just routine, but the disadvantage is that many elements in a EDP system are affected:

- structures, for example:
  - forms
  - data screens
  - temporary files
  - registers
  - reports, etc.
- programmes
- stored data
- documentation, among other things:
  - system descriptions
  - user manuals

the task rapidly becomes very extensive.

ALMOST IMPOSSIBLE TO CONTROL

Furthermore, it is such that changes in many elements are dependent on one another inasmuch as they actually must commence at the same time (on the same run). For example, alterations of one single date field in a register will lead, at the least, to the following changes in activities:

- changes of record structure
- conversion of actual register data together with possible back-up versions
- adjustment of all programmes using the field
- recompiling all programmes using the record
- adjustment of the system description

Conversion in 1 field at a time will mean that each register and every programme must be changed many times. Conversion of a record or a register at one time makes it difficult to maintain a comprehensive view. The risk that not all the necessary corrections have been done right is ever present and one should be prepared to be able to return to an earlier version.
In the case of large installations with hundreds of on-line data sets and thousands of programmes a conversion from 2 digit to 4 digit years will demand a very large control effort and the risk of errors, with following interruptions in operation, will still be considerable.

At Kommunedata each system is developed and maintained in one of five development centres, whilst operation is done at three regional central stations. Thus, there is an increased need for control and as far as Kommunedata is concerned the method of changing all date formats to 4 digit years will be almost impossible to control.

SELDOM A TRUE DOUBT AS TO THE CENTURY

Before starting with conversion of date formats, let's see if it is really necessary.

If at all times a date has a relevant range of less than (or equal to) 100 years, then a 2 digit year is theoretically adequate to identify the year within the relevant interval of time.

Example on conversion table with relevant interval = 1975 - 2015

2 digit year	75	87	00	15
Actual year	1975	1987	2000	2015

As end users know the relevant interval they will never be in doubt as to the meaning of the 2 digit year.

If the relevant interval can be greater than 100 years it is maybe not possible to make do with 2 digits in the year, but in 99% of such situations 4 digits will already be used in the year. Otherwise it may be necessary to convert such date fields - but the task will in any case be reduced to a fraction.

CHANGES IN PROGRAMMES ARE ADEQUATE

If information as to the valid time interval is added to the programme - maybe just the start year for the 100 year interval - the programmes will be able to handle all time calculations correctly. Information that the relevant interval starts in 1955 will for example, mean that 55 - 99 is interpreted as 1955 - 1999, whilst 00 -54 is interpreted as 2000 - 2054. The relevant time interval will often be definable relatively in relation to a run parameter or, the day's date field on a data screen. In this way a maintenance load is avoided in definition of the time intervals.

MINIMUM CONTROL

The advantages of only changing the programmes is that the number of elements in the conversion is drastically reduced and - maybe even more important - there is no need to coordinate the conversions. All programme changes are independent of one another. They just have to be effected before it is too late. This means the programmes can be corrected at a rate which fits in with other activities, that is it can be done at the same time as normal maintenance.

With this method of procedure control is minimal as one must just ensure that all programmes are made "bomb proof".

STANDARD ROUTINES ARE IMPORTANT

It is obvious the programme logics are made more complicated in relation to the present situation. Date manipulations are, however, already rather complicated so this added minor complexity is not so important.

The use of standard routines is a great help, it reduces both the manual effort as well as the risk of errrors. Already today, most date manipulations take place via standard routines and as more operations, which were previously very simple, become more complicated as we approach year 2000, the standard routine library should probably be extended in connection with demounting of the date bombs.

PROBLEM ANALYSIS IS NECESSARY

When one chooses to change only the programmes it is necessary to discover which operations in the programmes will go wrong in year 2000. Such an analysis is also a fine basis for judging which standard routines that should be offered to minimize the resource effort.

In the following we attempt to make a fairly comprehensive problem list.

DATE CALCULATION

By this we mean:

- number of days between two dates (or year, month and day)
- estimating from one date to a new date by stating the number of days between (or year, month and days)
- week day for stated date
- type of day (week or Sunday, Bank Holiday,etc.)
- conversion from one date format to another (for example on presentation of data screen, report or the like).

The above can easily be solved with 2 digit years and statement of relevant interval.

SORTING

Sorting of 2 digit years with usual utility programmes will result in erroneous chronological number order. Correct number order is achieved as follows:

- definition of sorting sequence for first digit in year (decade), for example 6789012345 for time interval 1960-2059. This method can only be used if the interval is at the start of a decade.
- temporary addition of auxiliary fields stating the century and included in the sorting criteria.
- a sorting routine which can sort with transposed zero point. If the transposition for a 2 digit number area is set at '60', the order of the sorting should be 60-99, 00-59.

A special problem presents itself if one has a 6 digit date included in the primary key of a file organised index sequentially. In many cases it will probably be advisable to convert to a 8 digit date - with the problems this involves.

'00' AND '99'

Many people know that '00 and '99' in a year have been attributed a special meaning as for example 'not filled in', 'first' or 'last'. If one had 4 digits in a year one would probaly have used '0000' and '9999' and thus been able to continue with this for a very long time. When 1999 and 2000 no longer are improbable or illegal years, can '99' and '00' on the other hand no longer be used and a new solution must be found.

In the case of date fields this problem probably does not exist except for clean year fields. '00' and '99' are still illegal values for months and days.

COMPARISON OF DATES

This problem is named separately, as many programmes test according to which of two dates is the latest. This is done apart from the standard routines as opposed to the more complicated date calculations.

Around year 2000 this test will become more complicated and must be re-programmed with consideration to the relevant time intervals or be done via new standard routines.

There may be other simple (at present) date manipulations which can be done apart from standard routines where the same considerations can be made. Please note though that the simple test of whether two dates are the same, will continue to work, as long as the relevant time interval is less than 100 years.

CALCULATION OF YEAR

Calculation with years is at the moment so simple that it is not done via standard routines.

With 2 digit years on both sides of year 2000 these calculations will become more complicated and it may be necessary to carry them out via new standard routines.

PRESENTATION OF DATES

That which gives problems with presentation of dates for the end user is that there usually is no doubt as to what extent 05 should be interpreted as 1905 or 2005. The problem is that there is more than one commonly used number order of year, month and day (at least in Denmark). Year - month - day (87-06-30) and day month year (30-06-87). After year 2001 doubt can, therefore, easily arise as to the meaning of 01-02-03. Is it 1st February 2003, or 3rd February, 2001?

Therefore, at some future point it may be necessary always to present dates with 4 digit years or, to state the format (yy-mm-dd).

Transcribing 4 digit years does **not** require storing or keying in 4 digits.

KOMMUNEDATA'S PLAN

After an analysis of the problem at Kommunedata a method or procedure has been chosen which is briefly as follows:

4-DIGIT YEARS WHEN DEVELOPING NEW SYSTEMS

Full information as regards the year is the most simple to handle and is set as a requirement in future developments. A complete set of standard routines is first constructed and made available for all non-trivial date manipulations.

Dating in forms and data screens can still use shortened years, if it is thought expedient, but stored years **must** contain 4 digits.

RUNNING ADJUSTMENT OF EXISTING PROGRAMMES

Gradually as programmes are changed as a link in the maintenance plan, adjustments and tests are likewise made to manage year 2000. Only if it is probable that they survive year 2000 though. Programmes which are tested and found (time) bomb safe, are specially marked and noted on a special list.

Also in this case a set of standard routines will first be made available to make it possible for statement of the valid time intervals. If this is not used, a standard interval which is made up-to-date every year or, every other year is used.

Standard routines also include conversion between various date formats - for example dates which are exported from an old system (shortened year) to a new system (full year).

SYSTEMATIC DEMOUNTING OF TIME BOMBS

In plenty of time prior to 2000 (1995-1997) a systematic inspection of the programmes not yet registered as "bomb free" is to be started. Maybe mechanical tools will be used to disclose which programmes use date fields.

The programmes which must survive year 2000, are adjusted by use of the new standard routines.

DECISIVE ADVANTAGE

When choosing the procedure outlined, Kommunedata stresses the fact that there is thereby achieved a number of really decisive advantages in relation to a slavish conversion of the date formats.

MINIMUM MANUAL RESOURCE EFFORT

The extent of the task is simply considerably reduced as it is only programmes which have to be altered. The extra logical complexity is unimportant.

MINIMUM RISK OF ERROR

As the individual programme correction can stand alone, the risk of error due to lack in coordination is avoided. Fewer operation stops again mean greater productivity. Return to an earlier version is easier with this procedure.

INDEPENDENCE

The lesser need for coordination is a great relief for a firm such as Kommunedata. This, together with the fact that programme changes can be fitted optimally in the remaining work, again means greater productivity.

ALSO ADVANTAGEOUS FOR OTHERS THAN KOMMUNEDATA

The problems are not unique for Kommunedata

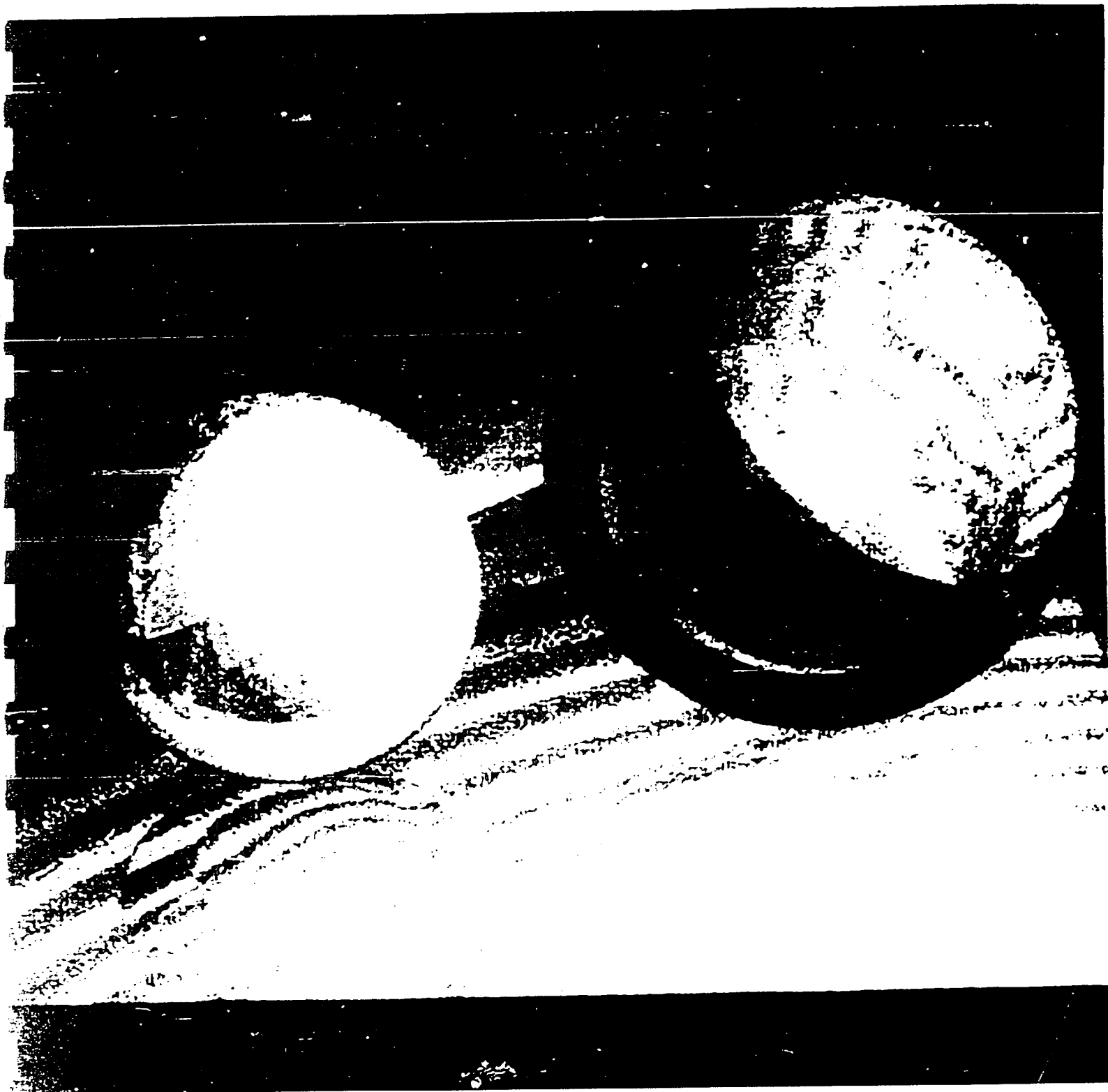
If these problems sound rather similar to the ones in your EDP installation, then I am certain there must be an advantageous solution close to the one chosen by Kommunedata for you.

Anyway, here is the suggestion at your free disposal.

IBM

Systems Journal

Vol. 25, No. 2, 1986



IBM
Systems Journal
Vol 25 No 2 1986

Computer processing of dates outside the twentieth century

by B. G. Ohms

This paper presents practical solutions to problems envisioned in extending computer processing of dates beyond the twentieth century. Many data processing managers are concerned with processing cross-century dates, and in doing so using existing systems, with a minimum of disruption to normal operations. The use of existing date formats can eliminate the need for massive system modifications. Methods of using existing date formats across century boundaries are explained. The use of a format termed the Lilian date format in honor of Luigi Lilio, the inventor of the Gregorian calendar, is introduced. The requirements for an effective date-processing algorithm are presented.

The Gregorian calendar serves us quite well in our day-to-day living. Due to discontinuities in various date divisions, however, it is not readily adaptable to computer programming. This fact becomes more apparent as we approach the new century. Few efficient, easy-to-use functions for manipulating dates have been produced. Also to be considered are the human requirements for ease of use, development, and maintenance. Other considerations include storage costs, efficiency, and adaptability across many different applications and environments.

Some early date-conversion programs were acts of expediency rather than planning, created to solve specific problems rather than for general programming use. Different functions were created at different times. Naming conventions, invocation formats, and implementation methods have often been inconsistent. Programs that provided a day-of-week function were rare. Also rare were programs for deriving a new date by adding or subtracting from another date in a different year. The functions that were available were normally not part of an integrated system for providing compatibility with most of the common date formats. Since documentation was not always created or maintained, individuals were often unaware of what was available. Today, dating format standards are being discussed internationally. The date-processing method presented in this paper is expected to be compatible with any foreseeable international standard date format as well as with the several formats discussed in this paper.

Typically, dates are displayed and stored in the *Julian* and *Gregorian* formats. Concepts and formats of both Julian and Gregorian date formats are discussed later in this paper. Simply stated, the Julian date is the number of the year together with the serial number of the day of that year. The Gregorian date format consists of month, day of month, and year. Many calendars show Julian dates printed in small numerals.

A format termed the *Lilian date format* is presented here as the basis for making date conversions of the

type mentioned earlier. The Lilian format, which may be stored in three bytes of memory, provides the storage capacity for dates well beyond the year 10 000. This format handles processing across century years and other aspects of date conversion not currently adaptable to computer programming. Practical date processing services must provide ease of use for the end user, developer, and maintainer. Such services must also eliminate date ambiguity, achieve excellent performance, and minimize storage.

The two positions traditionally used in both Julian and Gregorian date formats implicitly represent a year within a century. However, this system is inadequate for representing dates in more than one century. For example, it is ambiguous as to whether 03 represents the year 1903 or the year 2003. The Lilian date format avoids the ambiguity by using seven positions for the number of days from the beginning of the Gregorian calendar, October 15, 1582.

Origins of date complexity

Julian calendar. The Julian calendar was established in 46 B.C. by Julius Caesar.[1] It replaced a system of constant adjustments, more for political reasons than for correcting inaccuracies in timekeeping. After a bad start, with the first few leap years being calculated incorrectly, it served quite adequately for many centuries. The Julian calendar is not the same as the Julian date, which was previously defined. The Julian calendar was a time-measuring system, which we now discuss.

The scheme of the Julian calendar was quite simple. The calendar year consisted of 365-day years, with years evenly divisible by four having 366 days. The resulting average year of 365.25 days was slightly longer than the *tropical year*, which is based on the time marked by the passage of equinoxes. That slight difference resulted in a gradual drift of the calendar year. The resulting difficulty in properly setting the date of celebration of Easter caused great concern to the Roman Church. The date for Easter is related to the date for Passover, which is related to the vernal equinox. By the sixteenth century, the discrepancy approximated ten days, and the desire for calendar reform intensified. An artifact of this discrepancy manifests itself today in the differences in dates for Christmas and Easter between the Western Church and the Eastern Church. The latter continues to use the Julian calendar dates as they were at the time of the Gregorian calendar reform. That is, the Eastern Church recognizes the Gregorian calendar, but for certain feast days does not void the ten-day error that existed at the time of the calendar reform.

Gregorian calendar. For the calendar reform, Pope Gregory XIII selected[2] the plan of Aloysius Lilius (1510–1576?),[1] who is also known as Luigi Lilio.[3] This reform, known as the Gregorian calendar, was implemented on Friday, October 15, 1582.[2]

England and her colonies adopted the Gregorian calendar on Thursday, September 14, 1752.

Although use of the Gregorian calendar spread rapidly among Roman Catholic countries, many centuries passed before it was generally used by non-Catholic countries. England and her colonies, including what is now the United States, adopted the calendar on Thursday, September 14, 1752.[3] Many other countries—notably China, Greece, and Russia—did not adopt it until after the beginning of the 20th century. In fact, Russia adopted the Gregorian calendar on two separate occasions, first in 1918, about the same time as many other countries, and again on June 27, 1940.[4] Changes made in Russia in 1929 to avoid the religious associations of the Gregorian calendar were unsuccessful, and it was reimplemented in 1940.

The reform that synchronized the calendar year with the tropical year required the removal of ten days from the calendar. As a result, Friday, October 15, 1582, immediately followed Thursday, October 4, 1582. (An alternate plan would have removed the ten excess days gradually by canceling leap days for 40 years.) As before, every fourth year is a leap year, and, to maintain synchronization, centurial years that are evenly divisible by 400 are also leap years.[2] Thus, for example, 1900 was a common year; the year 2000 will be a leap year, and the year 2100 will start a series of common centurial years again. The result is an average calendar year of 365.2425 days, which closely approximates the tropical year. Despite

this close approximation, by the 44th century,[5] the Gregorian calendar will again differ from the tropical year by one day. Because the tropical year consists of an irrational number of days, no calendar year with an integral number of days can exactly match the tropical year. Not only is there not an integral number of days in a tropical year, but also the length of the day is not constant. That is, the length of the tropical year is also changing gradually.

Discontinuities in the Gregorian calendar scheme cause most of the difficulties with date manipulation.

Algorithms for date processing

Centurian date format. Discontinuities in the Gregorian calendar scheme cause most, if not all, of the difficulties associated with date manipulation. Instances of discontinuities are the following:

- No calendar unit is evenly divisible by weeks.
- The number of days per month varies.
- The number of days per year varies.
- The number of days per century varies.

These facts must be accounted for in calculating the number of days between two dates, calculating a date based on the addition or subtraction of days from another date, and calculating the day of the week for a date. A *centurian date format* (DDDDD, representing the day within a century) works well by giving continuous values within a century. Also, other date formats and day of week are readily calculable from this value. The centurian format is limited to a century and results in discrepancies when a century boundary is crossed. Consideration must be given to century years when making a leap year calculation. The DDDDD format will not span more than 273 years (or 179 years in a two-byte binary format). Also, a standard point of origin for the starting day must be defined.

Lilian date format. The Lilian date format consists of seven positions for the number of days from the beginning of the Gregorian calendar. Day one is Friday, October 15, 1582, and the value is incremented by one for each subsequent day. Based on this format are services supporting 44 experimental functions (or more if the three DMY, MDY, and YMD experimental versions of the Gregorian format are counted) synthesized from eight mnemonics.

Function-naming convention. Functions are named to promote easy recall and to suggest the activity to be performed. In the algorithm and experimental PL/I program discussed in this paper, each function name is synthesized from two 3-letter mnemonic parts. These mnemonic parts are defined as follows:

- CLL: compact Lilian date
- DAY: day of the week
- GRG: Gregorian date
- JUL: Julian date
- LIL: Lilian date
- SGR: short Gregorian date
- SJL: short Julian date
- VAL: validate a date

The first part is a description of the *target*, and the second part is a description of the *source*. VAL (validation) and DAY (day of week) can appear only in the target position of the function.

Arguments presented to any of the conversion functions are always presented in the same order: new date (target); old date (source); range date (control), when required; and Gregorian format [specifier(s)], when required. Table 1 illustrates the format for conversion arguments. In all instances, a return code is set.

The format descriptions in Table 1 refer to the type of data—D for day, M for month, and Y for year, as well as the number of positions (e.g., YY for two year positions)—that the data occupy. The origin of the term "Julian date" for the YYDDD format is given in Reference 6. Nevertheless, dates represented in the Julian format conform to the Gregorian calendar and not to the Julian calendar. The Julian date for Thursday, November 14, 1985, is 85318 (short form) and 1985318 (extended form).

Algorithms. The process of conversion between a Lilian format date and a Julian format date is now described. The algorithms are simpler to calculate by choosing a virtual base date rather than the real one. After the calculations are made, any discrepancies are resolved. In the following algorithms, the num-

Table 1 Examples of conversion function arguments

Argument	Description
target=JULSJL (source, 1925)	Conversion to a Julian (YYYYDDD) date from a short Julian (YYDDD) date within the range 1925–2024
target=JULGRG (source, DMY)	Conversion to a Julian (YYYYDDD) date from a Gregorian (DDMMYYYY) date
target=LILGRG (source, YMD)	Conversion of a Lilian (packed format) date from a Gregorian (YYYYMMDD) date
target=CLLJUL (source)	Conversion of a compact Lilian (binary format) date from a Julian date
target=SGRGRG (source, YMD, MDY)	Conversion of a short Gregorian (YYMMDD) date from a Gregorian (MMDDYYYY) date
CALL VALJUL (source)	Validation of a Julian date
target=DAYSJL (source 1925)	Day of week for a short Julian date

bers in parentheses are results of calculations made on the previously given date, November 14, 1985.

Extended Julian to Lilian. Given an extended Julian date (YYYYDDD), compute the corresponding Lilian date. First, compute the number of days from virtual January 1, 1501, to the start of the year being converted (1985318 Julian).

- Subtract 1501 from the Julian year (484).
- Multiply the difference by 365.25 (i.e., the average number of days per year) (176781.00).
- Truncate the result. The leap day is kept only for full four years (176781).

Then compute the number of Julian calendar (not Julian format) days from October 15, 1582, to the date being converted.

- Subtract 29872, i.e., the number of days between virtual January 1, 1501, and October 15, 1582 (146909).
- Add the Julian days (+318 = 147227).

Next, make the Gregorian calendar adjustments to this value.

- Subtract 1501 from the Julian year (484).
- Divide the difference by 100. One leap year is usually skipped each century (4.84).
- Truncate the result to obtain the number of whole leap days. Partial leap days do not exist (4).
- Subtract the number of whole leap days from the number of Julian calendar days (147223).

Because one out of every four centuries keeps all of its leap years, use the virtual year 1201, which is the beginning of the four-century cycle that contains the year 1582, to compute the number of leap years up to the target date.

- Subtract 1201 from the Julian year. The first four-hundred-year cycle began with virtual year 1201 and ended in the real 1600 (784).
- Divide the difference by 400 because one century leap year per 400 years is kept (1.96).
- Truncate the result because partial leap days do not exist (1).
- Add these leap days back into the adjusted Julian calendar days (147224).

This final result is the number of Gregorian calendar days from the beginning of the Gregorian calendar (October 15, 1582) to the date being converted. This result is the sought-for Lilian date.

Lilian to extended Julian. The purpose of this example is to show the procedure for converting a Lilian date to an extended Julian date. First, create a virtual Lilian date, with a starting point of January 1, 1201. Convert this result into a Julian calendar (not Julian format) date. Then convert the Julian calendar date to a Julian format date. The procedure is as follows (147224 Lilian):

- Add 139444 to the Lilian date. This is the number of days from virtual January 1, 1201 (the start of a 400-year cycle) to October 15, 1582. The result is a pseudo-Lilian date (286668).
- Divide the pseudo-Lilian date by 36524.25, the average number of days per century (7.85).
- Truncate the result to obtain the number of century leap days (7).

- Add the number of century leap days to the pseudo-Lilian date (286675).
- Divide the number of century leap days by 4 (1.75).
- Truncate the result to obtain the number of four-century leap days (1).
- Subtract the number of four-century leap days from the pseudo-Lilian date, because they are already included (286674).

The result is the number of full Julian calendar days from January 1, 1201, to the date being converted. We now convert this to an extended Julian format date.

- Divide full Julian calendar days by 365.25. This usually gives one less than the year for the date being converted (784.87).
- If there is no remainder from the division, subtract one from the quotient; otherwise, truncate the quotient to get the pseudo-Julian prior year (784).
- Multiply the pseudo-Julian prior year by 365.25 to determine the number of annual Julian calendar days prior to the year for the date being converted (286356.00).
- Truncate the number of annual Julian calendar days to eliminate partial leap days (286356).
- Subtract the number of truncated annual Julian calendar days from the full number of Julian calendar days to obtain the number of current Julian calendar days in the year of the date being converted (318).
- Add 1201 to the pseudo-Julian prior year to obtain the real Julian year, i.e., 1200 for years prior to 1200 plus one for the current year (+784 = 1985).
- Multiply the real Julian year by 1000 to put it into its proper Julian format position (1985000).
- Add the current Julian calendar days to the result to complete the Julian format date from the Lilian format date (1985318).

Ease of conversion

Accommodating end users. End users usually enter two digits for the year in a date and understand the ambiguity that this represents. Therefore, even at the turn of the century, to avoid adverse user reaction, programs must continue to function with only two digits for year. The inference of the year 1997 from 97 and 2003 from 03 must continue. For the exceptional case where the correct meaning could be 1897 and 1903, entry of all four digits may be required.

The month-day-year and day-month-year formats are ambiguous. Therefore, it might be advisable to continue presenting the date in its conventional U.S. format with a parenthetical explanation of the format—such as (MM/DD/YY)—to avoid the ambiguity.

It is not necessary to change date formats in files, because it is possible to change the programs only.

However, it may be necessary to provide a conversion function that receives a definition of the implied century as a parameter. An excellent way to do this unambiguously is to specify a year as the desired starting point of a 100-year range. For example, if the starting year for the range is specified as 1925, dates with year digits of 25 through 99 would be between 1925 and 1999, and dates with year digits of 00 through 24 would lie between 2000 and 2024.

Accommodating systems support. The conversion of isolated files to new date formats presents a rather trivial problem. In most cases, however, it is not possible to isolate the process. All programs that access the modified data must be changed simultaneously. In some large systems, literally thousands of programs may be involved. In these large systems, it may be prudent to avoid the cost and risk of massive changes in a short period of time.

It is not necessary to change date formats in files, because it is possible to change the programs only, so that the implied century in a date is recognized. Of course, in the vast majority of cases, that is exactly what does take place. Dates familiarly and implicitly exist within the 100-year range beginning with 1900 or 1901. Thus it is necessary merely to modify the programs so that the 100-year range starts at a later date. A beginning date set eighty years prior to the current systems date may be a reasonable convention. This is well within the range now in use.

The significant feature of this approach is that everything need not be modified at once. The modifications may be made over a period of years during

normal program maintenance. Of course, as systems are maintained or replaced, it would be practical to implement full information date formats. Where systems contain dates that span a range of more than 100 years, the century must already have been carried. In the rare event that this is not true, immediate conversion is unavoidable. Fortunately, in most applications, we can deal with centuries as exceptions rather than as a common problem.

Computational considerations

Storage. The two main considerations pertaining to storage are (1) the cost of storing large quantities of data; and (2) the computational cost of converting records within computer files to larger date-field sizes. When millions of dates are stored, as they are in most business systems, every additional byte required to save a single date multiplies to millions of additional bytes of storage. The programming necessary to accommodate larger date-field sizes in records further complicates date conversion.

Putting bounds on data-storage costs at reasonable levels and reducing conversion complications are both achievable. The allocation of additional space to record four positions for year, rather than the traditional two, is not the only possibility. Many systems currently store dates internally in packed Julian format, requiring three bytes of storage. In packed format, a Lilian date or an extended Julian date (YYYYDDD) both require four bytes of storage. However, if a binary format were to be adopted, either form of date could be stored in the three bytes used at present.

A more restrictive situation exists for systems in which dates are stored in two bytes instead of three. These systems use a binary format to record centurian or other forms of dates. In the near term for these systems, it may be necessary to continue storing dates in only two bytes. This cannot be accommodated with a Julian format. However, it is possible to store a range of dates by taking advantage of the continuous characteristic of a Lilian date.

Using the Lilian date system, any date in a selected range of 179 years may be stored as follows. Assume that the selected range is January 1, 1901, through December 31, 2079. Find the Lilian value of December 31, 1900. Subtract this value from the Lilian value of the date to be stored. Convert the result to a 16-bit (two-byte) binary value and store the result. Reverse the process to restore the two-byte date to Lilian format. Continuing to store dates in two bytes should be considered only where the programming cost of increasing record sizes in an existing system is prohibitive. The decreasing cost of storage makes the use of the full Lilian or Julian format a practical possibility when an application is being modified.

In the experiments on which this paper is based, the three-byte Lilian format for data storage has been found to be preferable. Internally, in computer use, the continuous nature of the Lilian date accommodates the types of processing that normally take place within a program. In addition, for all three storage sizes, a consistent format (i.e., the all-Lilian format or the quasi-Lilian format) is maintained.

The continuous nature of the Lilian date accommodates the types of processing that normally take place.

Flexibility. In our experiments, the IBM System 360/370 assembly language was used for coding the date functions. However, the method is easily adaptable to other programming languages such as COBOL, PL/I, and RPG. The experimental functions were also made re-entrant for flexibility within on-line environments.

Validity. Ideally, the validation of a date is necessary only at the point of its manual entry into a system. Experience teaches us that it is not unusual for an interfacing system to provide invalid dates. Therefore, all dates passed to conversion functions must be validated. When an invalid date is encountered by the experimental system, a null value is returned to the invoking program and a return code is set to indicate the nature of the invalid condition.

Efficiency. Date conversions are used heavily within certain applications. Because of the impact on a single system or an installation, efficiency must be inherent in date-conversion programs. Storage requirements for external display and internal calculations by computer programs are expected to mo-

tivate a high volume of conversions. Widely used programs in high-volume environments must perform well and not consume excessive amounts of storage. A date-conversion program that is good in all other ways will not be widely used if it is an operational bottleneck.

For the program discussed in this paper, written in System 360/370 assembly language and implemented as a set of PL/I functions, the experimental results indicated good efficiency. The longest execution paths, including PL/I prologue, validation, conversion, and PL/I epilogue, are a little more than one hundred machine-language instructions. Although the functions appear to the invoking PL/I program to be separate programs, they are all actually provided within a single program that requires less than 4K bytes of storage.

Experimental results indicated good efficiency.

Concluding remarks

Programmers require date-processing functions that effectively handle applications for both the present and the future. For the present, it is sufficient to validate source dates, to convert from one traditional date format to another, and to perform addition or subtraction operations involving dates. For the future, additional functions are required that support processing restricted only by the limitations of the Gregorian calendar. These functions must be fully compatible with existing date-format and record-size restrictions. Massive conversion efforts should not be required to process and store dates outside the twentieth century. Also, end users should be able to continue to use existing two-digit date formats when interacting with computer systems. In all cases the programs that provide these services must be reliable, efficient in the use of both processor and storage, and flexible in application.

Programs that embody all these qualities have been written and tested experimentally. The application as written requires less than 4K bytes of storage and has an average execution path of fewer than 100 machine language instructions. Re-entrancy and the possibility of multilanguage implementation indicate excellent flexibility. This approach presents a practical method of processing dates that is compatible with any dating format standard.

Acknowledgments

No significant work is done in isolation. Over the years, I have experienced the inspiration, assistance, and patience of many individuals. My colleagues Tom Gauthier and Jim Willard first sparked my interest in date processing several years ago. Recently Alex Chang, Barb Henderson, Jack Henriksen, Fred Lange, Bob Lord, Libby Ross, Karen Seabury, and Billy Shih have patiently served as a sounding board, made helpful suggestions, and have been active supporters. Sandy Mink provided valuable editorial support. Many unnamed others have been involved in my experiment, including every member of my family. Their support is also greatly appreciated.

Cited references

1. G. Moyer, "Luigi Lilio and the Gregorian reform of the calendar," *Sky and Telescope* 64, No. 11, 418–419 (November 1982).
2. J. J. Bond, *Handy Book of Rules and Tables for Verifying Dates Within the Christian Era*, Russell & Russell, a Division of Atheneum House, Inc., New York (1966).
3. *The New Encyclopedia Britannica*, Encyclopedia Britannica, Inc., Chicago (1980), p. 602.
4. F. Parise, Editor, *The Book of Calendars*, Facts on Life, Inc., New York (1982).
5. G. Moyer, "The Gregorian calendar," *Scientific American* 246, No. 5, 144–152 (May 1982).
6. M. A. Covington, "A calendar for the ages," *PC Tech Journal* 3, No. 12, 136–142 (December 1985). Joseph Justice Saliger (1540–1609) named the Julian date in honor of his uncle Julius.

General references

G. Moyer, "Astronomical scrapbook—Notes on the Gregorian calendar reform," *Sky and Telescope* 64, No. 12, 530–533 (December 1982).

International Organization for Standardization, "Writing of calendar dates in all-numeric form," Reference No. ISO 2014-1976(E) (April 1976).

International Organization for Standardization, "Information processing interchange—Representation of ordinal dates," Reference No. ISO 2711-1973(E) (January 1973).

Bruce G. Ohms *IBM Information Systems Group, 301 Merritt 7, Norwalk, Connecticut 06856.* Mr. Ohms joined IBM in 1967 and is currently a senior programmer/analyst working in the area of applications systems development. Prior to his current assignment, he has held a variety of positions in programming, systems design, analysis, and development center implementations. He received an A.A.S. degree in data processing from Belleville Area College, Belleville, Illinois, and he attended Yale University.

Reprint Order No. G321-5274.